Abstract

The amount of software in many systems increases exponentially. This increase impacts the reliability of these systems. In the source code of software many hidden faults are present. These hidden faults can transform into errors during the system life cycle, due to changes in the system itself or in the context of the system.

We will discuss the current trends and potential directions for future solutions of an increasing reliability problem.
Reliability of Software Intensive Systems

2 Gerrit Muller

version: 0.2
September 9, 2018
RSIScontent
Embedded Systems Institute (ESI)

**Mission**
To advance industrial innovation and academic excellence in embedded systems engineering (ESE).

**Vision**
ESI and its partners create and apply world-class ESE methods.

7 Founders:
- Industry (Philips, ASML, Océ)
- Universities (Twente, Delft, Eindhoven)
- Knowledge Institutes (TNO)

Reliability of Software Intensive Systems
version: 0.2
September 9, 2018
RSISwhatIsESI
Embedded Systems Engineering

- Performance
- Reliability
- Evolvability

In relation to all other system qualities: security, power, cost, size, et cetera
Industry as Laboratory

source of inspiration
application playground
industry

challenging problems
apply new engineering methods
research
evaluate
improve
hypothesis

observation
results

Reliability of Software Intensive Systems
version: 0.2
September 9, 2018
Gerrit Muller
Who is Gerrit Muller?
www.gaudisite.nl

deze lezing:

Exploration of Reliability

Introduction
  + ESI
  + Speaker

Exploration
  + trends
  + consequences for reliability
  + potential solutions

Research Projects
  + Trader
  + Tangram
  + Ideals
  + Boderc

conclusion
Trends in Embedded Systems

How to survive in innovative domains?

- Fast moving market
- Complex value chains
- Increased integration
- Fast moving technology
Increased Team Size

- **Historic Trend**
- **Our Challenge**

<table>
<thead>
<tr>
<th>Year</th>
<th>Required Team Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>10</td>
</tr>
<tr>
<td>2000</td>
<td>100</td>
</tr>
<tr>
<td>2005</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td></td>
</tr>
</tbody>
</table>
Number of Faults Proportional With Code Size

Based on average 3 errors/kloc

Reliability of Software Intensive Systems

Gerrit Muller

version: 0.2
September 9, 2018
ASMLproblem
The Hard Reset Syndrome: Power Down Needed!

Hard Reset Required:
- Cell Phone
- Television
- PC
- Beamer
- Car
- Coffee Machine
- DVD player
- Airbus

Pilot announces a flight delay, due to computer problems. A complete reset is required. The flight entertainment system also show a reset: a complete Linux boot. This reboot hangs: server xxx not found.
How to Make SW Intensive Systems Reliable

- Reduce functionality
- More procedures
- Reduce code
- Improve way-of-working
- More formalism
- More automation
- Early integration
- Shared understanding
- More feedback
- Tolerant interfaces
- Robust patterns

Within cost & time

Improve reliability

Without performance penalty

With increasing requirements

Increasing # suppliers, sites, et cetera

Improve robustness

Improve recovery

Improve detection

Improve recovery

Reliability of Software Intensive Systems

Gerrit Muller

version: 0.2
September 9, 2018
RSiSolutions
Reliability Research

Introduction
  + ESI
  + Speaker

Exploration
  + trends
  + consequences for reliability
  + potential solutions

Research Projects
  + Trader
  + Tangram
  + Ideals
  + Boderc

Conclusion
Example Trader Project

Overview

Trader   Television Related Architecture and Design to Enhance Reliability
Objective:  Develop method & tools to ensure reliable consumer electronics products
Research agenda:  System Reliability
Domain:  Digital TV
CIP:  Philips Semiconductors
Partners:  Philips CE, Tass, and Research DTI, IMEC, TUD, TU/e, UL, UT, ESI

Industrial Relevance

Poor reliability has severe business impact

- Customer expectation of TV reliability is high
  - Little tolerance for technical problems
- 100% fault-free design is not achievable
- High volume market implies high risks if reliability problems occur
  - Low product margin leaves no buffer for service costs
  - Service center costs multiplied by number of complaints
  - Market share reduction likely, i.e. customers buy another brand
- (On) Time to market is critical
  - Missing fixed shipping gates costs millions of dollars

User Perceived Reliability

- Objective
  - Determine the user-perceived severity of a product failure mode
- Methodology
  - Create a model considering relevant factors
    - User-perceived loss-of-functionality
    - User-perceived reproducibility
    - Failure-frequency
    - Work-around difficulty
    - User-group characteristics
    - Failure characteristics
    - ...
  - Validate model
  - Evaluate and suggest system failure recovery strategies
    - The recovery strategy may not annoy the user even more!

User Perceived Reliability

Aspects (all depends on user)  1  2  3  (JPEG)
Function importance  4  4  5
Frequency  4  1  5
Reproducibility  3  4  5
Solvability  3  2  5
Loss of Function / time / Behavior  4  5  0
Total:  576  160  0

Trader Domain Trends

TV complexity increase follows the PC world

from “Display movies over antenna”
to “Display anything over everything”

Broadcast
ATSC, DVB, ISDB, Analog
Terrestrial, Satellite, Cable

Connectivity
Cameras, JPEG, flash cards, HDD, MP3, Web browser, Ethernet, USB, etc...
Increasing Code Size in Televisions

1965
1 kB

1979
64 kB

2000
2 MB

1990
Moore's law

From: COPA tutorial, Rob van Ommering

Reliability of Software Intensive Systems

16  Gerrit Muller
Research: System Awareness to Improve Reliability

user input → system → system output

system

awareness

monitor

correction

customer expectations

system failure model
Expose product weaknesses at design time

- Source code analysis
  - Identify hotspots in code
  - Consider impact to user-visible behavior: prioritize warnings

- Software architecture reliability analysis
  - Techniques to identify failure-prone components
  - Evaluation of architectural alternatives and trade-offs
Quality Degradation Caused by Shit Propagation

- needed code
- bad code
- copy
- paste
- modify
- code not relevant for new function
- repair code
- new bad code
- new needed code

Reliability of Software Intensive Systems
19  Gerrit Muller
version: 0.2
September 9, 2018
BLOATshitPropagation
Example of Shit Propagation

Class Old:
  capacity = startCapacity
  values = int(capacity)
  size = 0
  
  def insert(val):
    values[size]=val
    size+=1
    if size>capacity:
      capacity*=2
      relocate(values, capacity)

Class New:
  capacity = 1
  values = int(capacity)
  size = 0
  
  def insert(val):
    values[size]=val
    size+=1
    capacity+=1
    relocate(values, capacity)

Class DoubleNew:
  capacity = 1
  values = int(capacity)
  size = 0
  
  def insert(val):
    values[size]=val
    size+=1
    capacity+=1
    relocate(values, capacity)
  
  def insertBlock(v,len):
    for i=1 to len:
      insert(v[i])
Example Tangram Project

Integration & Test

Early Integration

- Incremental
- Model Based
- Model ↔ Realization
- Infrastructure
- NDDS, Matlab, Chi

Results:
- Early Model Based Integration and Testing in AGILE
- Problem found in Sn CoMo before review

Test Selection & Sequencing

- 50% test time reduction (simulated)

Approach

System knowledge

Test sequencing

Test tree

Results: 10-20% Cycle Time Reduction in Goodsflow

Model Based Testing

Reliability of Software Intensive Systems
21 Gerrit Muller

version: 0.2
September 9, 2018
RSIStangram
Moore’s Law

Reliability of Software Intensive Systems
22  Gerrit Muller

version: 0.2
September 9, 2018
ASMLmooreLaw
Challenge: Exponential Increase

- Performance
- Feedback and adjustments
- HW and SW components
- Imaging overlay
- Productivity
- Reliability
- Robustness
- Innovation
- Complexity
4 Views on a Waferstepper

**subsystems**

- Laser light source
- Illuminator beam shaping
- Reticle stage positioning
- C&T contamination, temperature
- Lens projection
- Wafer stage positioning
- Wafer handler input/output

**control hierarchy**

- System control coordination
- Laser
- Illuminator
- Lens
- Measurement
- C&T
- Reticle stage
- Reticle handler
- Water stage
- Water handler
- Ethernet

**physics/optics**

- Laser
- Illuminator
- Uniformity
- Pulse-freq., bw, wavelength, ...
- NA aberrations transmission
- Lens
- Aerial image
- Sensor

**kinematic**

- Dynamic exposure through slit
- Vx, Vy expose, step, expose
- Wafer 250 mm/s
Dynamic simulation of integration & test approach:
1. Create model (modules, interfaces, faults, tests)
2. Execute model at any time
3. Balancing based on time, cost and remaining risk.

Balancing functionality, quality and time/cost
(over 20% reduction of integration & test time)
Example Ideals Project

code size \[\xrightarrow{\text{reduce by refactoring}}\] cross cutting concerns

Results: Parameter Check

- Automatic replacement of current parameter checking and tracing idiom with specific aspect oriented idiom.

- Code size reduction of 80% for parameter checking idiom (7% reduction of total module)
- Improved locality
Boderc Goal

Boderc goal =

A specific methodology to predict system performance within industrial constraints and restricted design space and analyze, discuss, document, and communicate throughput, quality, power computing response time, and predict people, process, project duration, and cost based on modeling multi-disciplinary.
Shared Understanding by Modeling

- 10^0
- 10^1
- 10^2
- 10^3
- 10^4
- 10^5
- 10^6
- 10^7

number of details

- mono-disciplinary
- multi-disciplinary
- system

- back-of-the-envelop
- formula based
- executable

Reliability of Software Intensive Systems
Gerrit Muller
Many Models Needed to Understand System

6. Kinematic modeling

7. Thermo modeling
8. Control architecture
9. Virtual printer models
10. Stepper motors

small, simple, goal-driven models
shorter cycle time, less cycles

shorter product creation lead time
Coverage of Reliability by ESI Projects

- within cost & time
- improve reliability
- without performance penalty
- with increasing requirements
- increasing # suppliers # sites, et cetera

- more/longer testing
- more procedures
- reduce functionality
- improve way-of-working
- reduce code
- improve recovery
- improve detection
- improve robustness
- more standardization
- improve recovery
- improve robustness

- more formality
- more automation
- early integration
- shared understanding
- more feedback
- tolerant interfaces
- robust patterns

- Ideals
- Trader
- Boderc
- Tangram
- Trader

Reliability of Software Intensive Systems
version: 0.2
September 9, 2018
RSIScoverage
Towards a Conclusion, Some more Trends

Introduction
  + ESI
  + Speaker

Exploration
  + trends
  + consequences for reliability
  + potential solutions

Research Projects
  + Trader
  + Tangram
  + Ideals
  + Boderc

Conclusion
Applications depend on chain of systems

users

infotainment appliance

watch video
browse photo's

calendar
and much more...
Interoperability: systems get connected at all levels

Reliability of Software Intensive Systems

version: 0.2
September 9, 2018
DYOFscopeOfInteroperability
Multi-dimensional interoperability

integrating **multiple** applications
clinical analysis
clinical support
administrative
financial
workflow

in **multiple** languages
USA, UK,
China, India,
Japan, Korea
France, Germany
Italy, Mexico

delivered by **multiple** vendors
Philips
GE
Siemens

based on **multiple** media, networks
DVD+RW
memory stick
memory cards
bluetooth
11a/b/g
UTMS

and **multiple** standards
Dicom
HL7
XML

and **multiple** releases
R5
R6.2
R7.1
### Inteopearability Trends and Research Challenges

<table>
<thead>
<tr>
<th><strong>trends</strong></th>
<th><strong>(partial) solutions</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>market dynamics</td>
<td>globalization</td>
</tr>
<tr>
<td></td>
<td>hype waves</td>
</tr>
<tr>
<td></td>
<td>Moore's law</td>
</tr>
<tr>
<td>interoperability</td>
<td>emerging behavior</td>
</tr>
<tr>
<td></td>
<td>future vs legacy</td>
</tr>
<tr>
<td></td>
<td>heterogeneous vendors</td>
</tr>
<tr>
<td></td>
<td>dynamics (continually changing)</td>
</tr>
<tr>
<td>reliability</td>
<td>complexity</td>
</tr>
<tr>
<td></td>
<td>heterogeneity</td>
</tr>
<tr>
<td></td>
<td>dynamics</td>
</tr>
<tr>
<td></td>
<td>#engineers involved</td>
</tr>
<tr>
<td></td>
<td>economical interests</td>
</tr>
<tr>
<td></td>
<td>latency due to slow process</td>
</tr>
<tr>
<td></td>
<td>most fundamental solution</td>
</tr>
<tr>
<td></td>
<td>in system feedback</td>
</tr>
<tr>
<td></td>
<td>human-in-the-loop feedback</td>
</tr>
<tr>
<td></td>
<td>semantic understanding</td>
</tr>
<tr>
<td></td>
<td>containment</td>
</tr>
<tr>
<td></td>
<td>gateway</td>
</tr>
<tr>
<td></td>
<td>....</td>
</tr>
</tbody>
</table>

*new research challenges!*
Conclusion

Based on average 3 errors/kloc, less code reduces the typical amount of errors per product. This in turn reduces the impact of hidden faults on system reliability. The graph shows a trend where more years of development and LOC (lines of code) per product correlate with a higher number of errors/kloc. From 1990 to 2005, the number of errors/kloc decreases, indicating improved development practices and quality assurance measures.

Reliability of Software Intensive Systems
version: 0.2
September 9, 2018
Gerrit Muller